The Magic of Algorithm Design and Analysis

Teaching Algorithmic Skills using Magic Card Tricks

João F. Ferreira
School of Computing
Teesside University
Middlesbrough, UK
HASLab/INESC TEC
Universidade do Minho
Braga, Portugal
joao@joaoff.com

Alexandra Mendes
Faculty of Arts
York St John University
York, UK
a.mendes@yorksj.ac.uk

ABSTRACT

We describe our experience using magic card tricks to teach algorithmic skills to first-year Computer Science undergraduates. We illustrate our approach with a detailed discussion on a card trick that is typically presented as a test to the psychic abilities of an audience. We use the trick to discuss concepts like problem decomposition, pre- and post-conditions, and invariants. We discuss pedagogical issues and analyse feedback collected from students. The feedback has been very positive and encouraging.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computers and Information Science Education—*Computer science education*; F.1.0 [**Computation by abstract devices**]: General

Keywords

Algorithms; Algorithmic Problem Solving; Invariants; Pre/Post-conditions; Hoare Triples; Magic Card Tricks; Puzzles and Games

1. INTRODUCTION

There is a growing number of educators who advocate the use of recreational problems and so-called "unplugged" activities to teach computer science concepts [2, 4, 12]. We have ourselves been using "unplugged" activities for a few years to teach algorithmic problem solving at undergraduate level [7, 8]. Recently, inspired by the engaging work done in the project cs4fn [4, 5, 13, 14], we incorporated magic card tricks into our activities.

In this paper, we describe our experience using magic card tricks to teach algorithmic skills to first-year Computer Science (CS) undergraduates. Our main contribution is to show how more formality can be added to the presentation of card tricks so that they can be presented at university level (this was suggested as future work in [4]). We enrich the contribution by including some pedagogical comments and by discussing feedback provided by the students.

We start in Section 2, where we discuss in detail a card trick that is typically presented as a test to the psychic abilities of an audience. The trick was taken from [14], a booklet on magic tricks that has been used to demonstrate computer science concepts to school students. After we describe the trick, we present a detailed explanation of why it works, highlighting the algorithmic techniques used along the way. In Section 3 we discuss some pedagogical considerations that make the teaching of algorithmic skills using card tricks more effective. We have collected feedback from first-year CS undergraduates who attended a learning session based on the trick. We analyse the feedback in Section 4. In Section 5 we present related work and we conclude the paper in Section 6, where we also discuss some of the next steps.

2. ALGORITHMIC CARD TRICKS

Algorithmic card tricks (also called self-working tricks) are tricks that do not involve any hidden mechanisms like rigged decks or double lifts. In other words, if the spectator replicates the *visible* steps performed by the magician, the same (surprising) result will be achieved. Essentially, we can describe algorithmic card tricks as algorithms that manipulate cards. We believe that card tricks are excellent vehicles to teach important algorithmic skills, because:

- Students think about the underlying algorithms on a more abstract level, ignoring implementation and computer language details;
- The kinaesthetic and interactive nature of manipulating cards helps students visualise the algorithms: they can quickly verify properties and test new hypotheses using the cards;
- Students learn that algorithmic problem solving applies to areas outside computer science;
- The recreational nature of card tricks promotes student engagement;
- Card tricks can be used to illustrate important concepts like loops and invariants, assertions, and problem decomposition.

2.1 Example: are you psychic?

In this section, we present a card trick whose analysis and proof of correctness is non-trivial. The trick is typically presented as a test to the psychic abilities of an audience. We analyse the underlying algorithm and show how the card trick can be used to practise problem decomposition and the identification of pre- and post-conditions. The correctness of the algorithm is proved by identifying and formulating two invariants. To describe the trick, we follow the explanation found in [14] (but we include more graphical aids).

2.1.1 Description of the card trick

We start by getting five pairs of cards out of the pack. Any five pairs can be used, but for simplicity, let us choose the five pairs 2♥ 2♠, 3♥ 3♠, 4♥ 4♠, 5♥ 5♠, and 6♥ 6♠.

Accepted version by ITiCSE'14.

Next, set the cards up in one ordered pile as shown in Figure 1. Show the pile to the audience and explain that the trick is about to start.

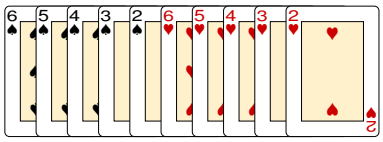

Figure 1: Five pairs of cards as one ordered pile.

To facilitate the explanation of the trick, we explain how to perform it with the faces of the cards up. *However, when performing the trick, the faces should be down!*

Spread the cards in your hand and have the audience point to any card. Split the pack at that point and place the top pile at the bottom of the pack. Repeat this until the audience is happy the cards are well mixed. For example, if the audience points to the four of hearts shown in the left image of Figure 2, the resulting deck is the one on the right image.

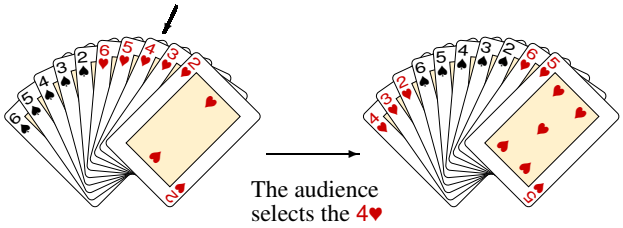

Figure 2: Mix the cards until the audience is happy, by cutting the pack at the point selected by the audience.

When the audience is happy the cards are well mixed, deal the top five cards, one at a time, into a pile on the table, thereby reversing their order. Place the remaining undealt cards in a second pile beside them. By putting this second pile straight down you have kept their order the same. For example, from the deck shown in the right image of Figure 2, we get the two piles shown in Figure 3. Please note that the card 5♥ is the top card in the right deck shown in Figure 2. (Remember that when performing the trick, the cards should have their faces down.)

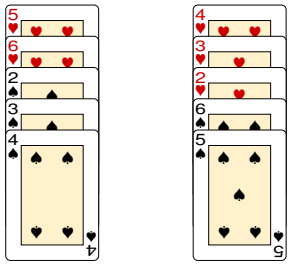

Figure 3: Division into two piles. The left pile contains the top 5 cards of the right deck shown in Figure 2, in reverse order. The right pile contains the remaining cards, in the original order.

Explain to the audience that as there are 5 cards in each pile you will give them 4 chances to use their psychic powers. They can have 4 swaps. A swap involves taking the top card on one of the piles and placing it on the bottom of the same pile. Explain that they can, for example, do all 4 swaps on one pile, 2 on each, or 3 on one and 1 on the other. It is their choice, remembering that their aim is to be left with two matching cards. For example, if the audience chooses to perform 3 swaps on the left pile of Figure 3 and 1 swap on the right pile of Figure 3, the resulting piles are the ones shown in Figure 4.

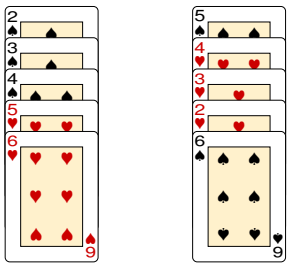

Figure 4: Resulting piles if the audience chooses to perform 3 swaps on the left pile of Figure 3 and 1 swap on the right pile of Figure 3.

Once the 4 swaps are made, remove the top card on each pile and place them aside (with their faces down!). Point out that it does not matter what they are as they are being discarded. Now there are 4 cards in each pile, as shown in Figure 5.

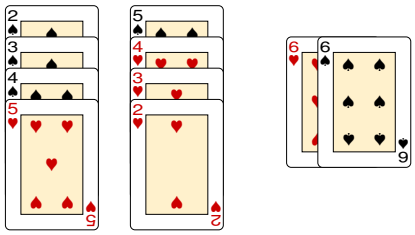

Figure 5: After the top card on each pile is removed, we are left with two piles of 4 cards. We place the two removed cards aside.

At this point, offer the spectator 3 swaps in total, and once the swaps are done remove the top two cards from the piles. There are now three cards left in each pile, so give them two swaps this time, and again remove the top card from both piles. This leaves two cards in each pile. This is their final chance to get it right. Tell the audience that one swap is left, and one card can make all the difference. They choose their swap, and the top two cards from each pile are discarded. Now it is time to reveal the final two single cards left on the table. **They match!** The audience chose freely how to mix the cards and which cards to eliminate, so it was their secret psychic powers that came through to ensure a match at the end. Give their jaw time to drop, then dramatically reveal that all the pairs of cards they removed match in value too.

2.1.2 Discussion and analysis of the trick

This trick can be used to discuss the specification of algorithms (via Hoare triples), the identification of pre- and post-conditions, problem decomposition, and the identification and formulation of invariants [8]. It also provides a good example to discuss the concept of *proof of correctness*, because students are very keen on understanding why the trick works.

In what follows, we explain our approach when presenting the analysis of the trick to first-year CS undergraduates.

Formal specification and abstraction. The first step in the analysis of the algorithm is to specify what it does. Using a *Hoare triple*, a high-level specification can be written as:

{ 10 cards: 2♥ 3♥ 4♥ 5♥ 6♥ 2♠ 3♠ 4♠ 5♠ 6♠ }

perform card trick

{ 1 pair: pile1 = N ∧ pile2 = N

where N is the number on a card }

In the description of the trick we referred to both piles as "left" and "right". Here, we name them pile1 and pile2. The expression "pile1=N ∧ pile2=N" means that pile1 and pile2 both contain one card with the number N; we use the conjunction operator ∧ to denote "and". The expressions between curly brackets correspond to assertions. An expression of the form { P } S { Q } where P and Q are predicates and S is a program statement (i.e., an algorithm) is called a *Hoare triple*. It means that if P is true before the execution of the statement S, execution of S is guaranteed to terminate in a state where property Q is true. We are stating that when we start the trick with the 10 cards 2♥ 3♥ 4♥ 5♥ 6♥ 2♠ 3♠ 4♠ 5♠ 6♠ we are always guaranteed to terminate with 1 pair of cards with the same number.

An important algorithmic skill is the avoidance of unnecessary detail, so we start by observing that the suits of the cards are irrelevant and the trick works with any five pairs. We can thus abstract from the specific cards used and be more general by introducing variable names to represent arbitrary cards:

{ 10 cards: $ABCDE\ ABCDE$ }

perform card trick

{ 1 pair: pile1 = N ∧ pile2 = N

where N is the number on a card }

There are many (and possibly better) choices for expressing the pre- and the post-condition. The one we use was tested and works well with first-year CS undergraduates.

Problem decomposition. Now that we have a high-level specification, a reasonable step is to decompose the problem into simpler components. The algorithm can be decomposed into three main parts: the "shuffling" process, where the audience chooses how to mix the cards; the division into two different piles; and finally, the part where multiple swaps are performed. We can express this decomposition as follows:

{ 10 cards: $ABCDE\ ABCDE$ }

shuffle

{ ? };

divide into two piles

{ ? };

perform swaps

{ 1 pair: pile1 = N ∧ pile2 = N

where N is the number on a card }

The goal now is to characterise the assertions marked with "?". We investigate each part separately.

The specification and decomposition of the problem are obtained through an interactive discussion with the entire classroom. For the remaining steps of the analysis, the students work together in small groups.

Shuffling step. In the first part of the algorithm, we ask the audience to point to any card and we split the pack at that point, placing the top pile at the bottom of the pack. We repeat this until the audience is happy the cards are well mixed. Figure 2 shows how the initial pack changes when the audience selects the four of hearts. The figure also shows that the cards are indeed mixed, that is, the order of the cards changes. However, there is a key property being maintained by this mixing process: *the top five cards match the bottom five cards*. In other words, the sequence of numbers corresponding to the top five cards is the same as the sequence of numbers corresponding to the bottom five cards. This can be easily observed in the right image of Figure 2, where the top five cards are 5♥ 6♥ 2♠ 3♠ 4♠ and the bottom five cards are 5♠ 6♠ 2♥ 3♥ 4♥.

This property is also satisfied by the initial configuration (shown in the left image of Figure 2). Since the mixing process corresponds to a finite sequence of rotations, the property will clearly be maintained. In other words, it does not matter how many times the pack is rotated, the top five cards will always match the bottom five cards.

We can thus conclude that an invariant of the mixing process is: *the top five cards match the bottom five cards*. A simple way to specify this invariant is as follows:

{ 10 cards: *the top five cards match the bottom five cards* }

shuffle

{ 10 cards: *the top five cards match the bottom five cards* }

We can avoid the use of natural language and be more consistent with the pre-condition written above, by introducing five new variables A',B',C',D',E':

{ 10 cards: $ABCDE\ ABCDE$ }

shuffle

{ 10 cards: $A'B'C'D'E'\ A'B'C'D'E'$ }

The introduction of new variables is necessary because we do not know what will be the first card in the sequence; nevertheless, we are certain that the top five cards will match the bottom five.

Creating two piles. When the audience is happy the cards are well mixed, we deal the top five cards, one at a time, into a pile on the table, thereby reversing their order. We then place the remaining undealt cards in a second pile beside them. By putting this second pile straight down we keep their order the same. For example, from the deck shown in the right image of Figure 2, we get the two piles shown in Figure 3. It is easy to see that these piles are reverses of each other. A simple way to specify this is:

{ 10 cards: $A'B'C'D'E'\ A'B'C'D'E'$ }

divide into two piles

{ 10 cards: pile1 = $E'D'C'B'A'$ ∧ pile2 = $A'B'C'D'E'$ }

A more compact and formal way of writing the post-condition is as follows:

{ 10 cards: $A'B'C'D'E'\ A'B'C'D'E'$ }

divide into two piles

{ 10 cards: pile1[k] = pile2[4−k], for $0 \le k \le 4$ }

The expression pile1[k] denotes the k^{th} card of pile1. We start counting at 0, so pile1[0] corresponds to the top card of pile1. Using this notation and naming the piles in Figure 3 as pile1 and pile2, we observe, for example, that pile1[0]=**4**=pile2[4 − 0] and pile1[4]=**5**=pile2[4 − 4].

Performing swaps. The last step of the algorithm consists of an iterative process, where a number of swaps are done and the size of the two piles is reduced until the piles only have one card each. We start with two piles of the same size (that are reverses of each other) and we keep reducing the size of both piles by 1. Initially, in the first iteration, each pile contains 5 cards, so the audience has to perform 4 swaps. One card is removed from each pile, meaning that in the second iteration each pile contains 4 cards and the audience has to perform 3 swaps. More generally, if at a given iteration each pile contains n cards, the audience has to perform $n-1$ swaps. This can be modelled more precisely (but still at a high-level) as follows:

{ 10 cards: pile1[k] = pile2[4−k], for $0 \le k \le 4$ }
swaps := 4 ;
do swaps ≥ 1 →
Ask for number j where $0 \le j \le swaps$;
Swap j cards in pile1 ;
Swap $swaps-j$ cards in pile2 ;
Remove the top card from each pile ;
swaps := *swaps*−1
od
{ 1 pair: pile1 = N ∧ pile2 = N
where N is the number on a card }

We write *do swaps* ≥ 1 → ··· *od* to express a loop that executes while *swaps* ≥ 1. So, the loop terminates (and the trick ends) when no more swaps can be performed. Let us focus now on the step where cards are swapped. The step "*Swap j cards in* pile1" puts the card pile1[j] at the top of pile1, without affecting the relative order of the other cards. Similarly, the step "*Swap swaps*−*j cards in* pile2" puts the card pile2[*swaps*−j] at the top of pile2, without affecting the relative order of the other cards. As a result, after the swaps, the top cards on pile1 and pile2 are, respectively, pile1[j] and pile2[*swaps*−j]. We can easily observe this in the transition from Figure 3 to Figure 4. In Figure 3, we have pile1[3] = 6♥ and pile2[1] = 6♠. The two piles shown in Figure 4 are formed after the audience chooses to perform 3 swaps on the left pile (pile1) of Figure 3 and 1 swap on the right pile (pile2) of Figure 3. We can see that the cards 6♥ and 6♠ are now at the top; moreover, the relative order of the other cards is maintained.

Given that the initial value of *swaps* is 4, we conclude from the pre-condition that pile1[j] = pile2[*swaps*−j]. As a result, after the swaps are performed the top cards of both piles form a pair. So, after the step "*Remove the top card from each pile*", a pair is removed and the resulting piles are reverses of each other. This means that an invariant of the loop is the property that the piles are reverses of each other. Given that the value of *swaps* is decreased, we can formulate the invariant as

pile1[k] = pile2[*swaps*−k], for $0 \le k \le swaps$.

As discussed before, the invariant is valid initially when each pile contains 5 cards. We have just discussed that the invariant is valid after each iteration, when the size of each pile decreases by 1. Therefore, the invariant will be valid on termination when each pile contains 1 card. So, on termination we will have two piles with one card each that are reverses of each other. This means that the two remaining cards must match! More formally, we observe that the final value of *swaps* is 0, so we can conclude immediately from the invariant that pile1[0] = pile2[0].

Further discussion. The formulation of the last step can be simplified by not introducing the variable *swaps*, because *swaps* can be determined by the number of cards in the piles. However, when presenting the trick, we feel that the introduction of variable *swaps* facilitates discussion; it also provides the opportunity to discuss program transformation: we often ask the students how can we rewrite the algorithm without using that variable.

As mentioned above, we target first-year CS undergraduates. For that reason, we avoid formalisms that students are not familiar with. If we were teaching more advanced students, we could use modular arithmetic to express the key properties of the algorithm and write more formal proofs of correctness.

3. PEDAGOGICAL COMMENTS

Although, in our experience, the recreational nature of card tricks naturally promotes student engagement, learning sessions still need to be planned considering psychological constraints on learning. In particular, sessions are more effective acknowledging that a) the attention of students is typically maintained for about 10 to 15 minutes, after which learning drops off rapidly; b) a change of activity every 15 minutes restores performance almost to the original level; and c) a period of consolidation at the end of the session greatly enhances retention [3].

We observed that the introduction of group work in the sessions on magic tricks contributes to their effectiveness. It is known that group work promotes active involvement and deep learning, mainly because it increases the amount of time that students spend thinking about conceptual ideas [16], encouraging discussion and negotiation of ideas and meaning. As stated in [17], *"having to achieve practical outcomes as a group can lead to more understanding of processes due to having to plan explicitly, articulate and agree the next steps forward"*. Moreover, an important aspect of group work is that *reflective aspects are sharpened because students readily identify each other's learning in a way they do not with top-down teacher-directed learning* [3]. As part of a formal peer-review system, one of the sessions where we present the card trick described in the previous section was observed by another academic member of staff. In her written comments, the observer confirmed that engagement was positive: "*Good student involvement and engagement from the start [...]*" and "*From where I was sitting (at the back) all students appeared engaged in the task*".

The observer also wrote that "*students responded well to questions*". Our sessions are normally structured around questions (e.g., it is very common to start the discussion with the question "How can we specify this algorithm?" or "How can we decompose the problem into simpler and smaller problems?"). As [16] suggests, this gives the students the opportunity to exercise responsible choice in the method and content of study. Moreover, because often students suggest multiple correct ways of tackling the problem, they decide which path to follow; this gives them ownership of the learning process. Normally, the questions that we ask are *convergent*, i.e., there is a correct (or "best") answer in mind and students are steered towards that answer. This promotes *social construction of knowledge*, where learners *contribute and agree on the structure as it emerges* [3].

4. EVALUATION

We have collected feedback from students to gain a better understanding of their opinion about the use of card tricks for teaching algorithmic skills. As part of a first-year undergraduate module on algorithms and data structures offered at Teesside University, we delivered a one-hour session on the card trick presented in Section 2. All the students enrolled in the module are studying for a BSc in Computer Science. Forty (40) students attended the session. They had the opportunity to work in small groups of two or three students. Each group had one deck of playing cards to replicate the trick.

The feedback was collected through an online, voluntary questionnaire that was completed by 23 students, in their own time outside the classroom. The questionnaire was made of six compulsory and three optional questions. Table 1 shows the first six questions and the average of the results on a Likert scale from 1 to 5, where 1 corresponds to *Strongly Disagree* and 5 corresponds to *Strongly Agree*.

The feedback is positive for all questions: the lowest average score is over 4.2 and the overall average score is a very encouraging 4.4. For questions 1, 4, and 6 there were no negative answers. This clearly suggests that students enjoyed the session and that they would like to have more sessions where card tricks are used to illustrate algorithmic concepts. It is also interesting to see that students found the use of real playing cards helpful to understand the algorithm. For questions 2 and 3, only one student disagreed with the statements. Another student disagreed with the statement in question 5. All other feedback for questions 2, 3, and 5 was either neutral or positive indicating that, although there is still room for improvement, the vast majority of students found the use of a card trick engaging, motivating, and a good way to learn and practise algorithmic skills. It is interesting to note that no student "Strongly disagreed" with any of the statements.

These results are encouraging and a good indication that card tricks are a good vehicle to teach important algorithmic skills.

The remaining three optional questions were more general:

1. *What did you like about this session? (If anything)*
   (19 answers) Several students indicated the interactivity of the session as one of the highlights: "*I enjoyed the interaction, by using objects to explain algorithms*". Some students indicated that the session was entertaining and unique: "*The uniqueness of the session. Nothing else done in this interesting manner*". Being able to "visualise" the algorithm through the card trick seems to be appreciated by several students. One student wrote "*Much more easier as I was able to see exactly how it worked instead of thinking how it worked and trying to get my head around previous tasks*". Other answers regarding the visual aspect of the session included "*The engagement within the lecture, and particularly the visualisation in which helped me understand the algorithm a lot better [sic]*", and "*I like the way that I could see the algorithm working*". Another student wrote "*I liked how well the use of the card trick was explained as the lecturer went through the steps clearly while demonstrating it in a practical way*". One student answered that he liked "*Everything*". In addition, feedback indicates that card tricks can benefit students with certain learning difficulties: "*I liked how I could visually engage with the lecture, having an auditory learning difficulty sometimes affects my ability to fully engage with new material so having a visual aid and something I could use with my hands helped me a lot*".

2. *What did you dislike about this session? (If anything)*
   (10 answers) 7 answers did not point out anything wrong with the session, most answering "*Nothing*" or "*N/A*". One of these 7 students wrote "*I didn't dislike anything really, it helped me understand algorithms a lot more*". One student indicated that it "*Would be nice if there were more cards*"; we cannot be certain, but the student was possibly indicating that each student should be given one deck cards (instead of one deck per group). One considered the session "*a bit fast-paced*". Another one wrote an answer unrelated with the content of the session.

3. *Please provide any further comments or suggestions that can be used for improving this session. (If any)*
   (5 answers) Two students indicated that more hands-on demonstrations could be included in future sessions. One of these students wrote "*More hands on as it makes it easier to understand the problem much quicker*". The other one stated that other puzzles used throughout the module could also be demonstrated in a similar visual way. Another student wrote "*I wouldn't just use card tricks. Keeping things different and interesting seems to help motivate people in my opinion*". Two students did not provide suggestions, answering "*None*" and "*N/A*".

Overall the feedback was very positive and clearly the session was a positive experience for the students. All suggestions for improvement include a more frequent use of similar approaches to teaching.

5. RELATED WORK

The current leading work in this area is the cs4fn project [5], whose goal is to enthuse school students about computer science and teach advanced computing ideas. The project consists of a free magazine, live interactive shows and a popular webzine. More details about their work with card tricks can be found in the booklets [13, 14] and in the papers [4, 5]. The work presented in this paper was developed after attending a cs4fn presentation where the card trick "Are you psychic?" was shown. We enjoyed it so much that we decided to add more formality to the presentation of the trick and test it with our first-year undergraduate students. We also extended the presentation to include other algorithmic concepts, such as pre- and post-conditions and problem decomposition. In [4], the authors write that their "target audience has been school students but the approach, with more formality added, could also be used to illustrate theory to university students too. We leave this as further work.". Our work contributes towards this suggestion.

Other related work includes the CS Unplugged project [2], who were pioneers in using magic to teach computing and algorithms. In [11], a trick for demonstrating binary numbers is shown and in [9, 10] a variety of tricks are used to demonstrate topics that include algorithms, modular arithmetic, and binary encoding.

A related line of work is the use of recreational problems to teach computer science. In [12], for example, the authors advocate a wider use of recreational problems in teaching design and analysis of algorithms. In [6], the authors introduce a sample syllabus and course material for engineering and computer science, using a puzzle-based learning approach; the main book on this approach is [15]. In [1], the author presents a problem-based approach to algorithmic problem solving, where all the problems have a recreational flavour. A similar approach is followed in [8], where principles and techniques of algorithmic problem solving are exemplified using recreational problems.

Table 1: Questions and scores

#	Question	Score
1	I enjoyed the session on "The Algorithmics of Card Tricks" (Lecture 11), where we analysed the algorithm behind a card trick	4.52
2	The use of a card trick improved my engagement during the lecture	4.43
3	I think that card tricks are a good way to learn and practise algorithmic skills	4.43
4	I would like to have more sessions where card tricks are used to illustrate concepts on algorithm design and analysis	4.30
5	The use of a card trick made me feel more motivated and interested in learning more algorithmic skills	4.26
6	Using real playing cards to simulate the card trick helped me understand better the underlying algorithm	4.43

6. CONCLUSION

Magic card tricks can be used to illustrate important algorithmic skills. We have shown how we use a specific card trick to teach first-year CS undergraduates concepts like problem decomposition, pre- and post-conditions, and invariants. We have discussed some pedagogical issues and we have analysed feedback collected from students.

We started this project inspired by the excellent work reported in [4], where the authors write that their "target audience has been school students but the approach, with more formality added, could also be used to illustrate theory to university students too. We leave this as further work.". Our work contributes towards their suggestion and the feedback that we collected suggests that card tricks can indeed be used to teach algorithmic concepts at first-year undergraduate level.

We intend to further develop the work shown here by adapting more tricks to teach first-year undergraduates. We also plan to test a more formal approach with rigorous mathematical proofs with students at a more advanced level. In both cases, we will perform more evaluation on whether students truly understand Hoare triples, and pre- and post-conditions.

ACKNOWLEDGEMENTS

We would like to thank all the students who provided feedback. We are also grateful to Eudes Diemoz for his encouragement and to the anonymous referees for their valuable comments.

REFERENCES

[1] Roland Backhouse. *Algorithmic Problem Solving*. John Wiley & Sons Ltd., 2010.

[2] Tim Bell, Ian H. Witten, and Mike Fellows. *Computer Science Unplugged*. 2010. Available at `http://csunplugged.org`. Last accessed: 17 Mar 2014.

[3] John Biggs and Catherine Tang. *Teaching for Quality Learning at University: What the Student does (Society for Research Into Higher Education)*. Open University Press, 4th edition, 2011.

[4] Paul Curzon and Peter McOwan. Teaching formal methods using magic tricks. In *"Fun with formal methods" at the 25th International Conference on Computer Aided Verification (CAV 2013)*, St Petersburg, Russia, July 2013.

[5] Paul Curzon and Peter W. McOwan. Engaging with computer science through magic shows. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, pages 179–183, New York, NY, USA, 2008. ACM.

[6] Nickolas Falkner, Raja Sooriamurthi, and Zbigniew Michalewicz. Puzzle-based learning for engineering and computer science. *Computer*, 43(4):20–28, 2010.

[7] João F. Ferreira, Alexandra Mendes, Alcino Cunha, Carlos Baquero, Paulo Silva, L.S. Barbosa, and J.N. Oliveira. Logic training through algorithmic problem solving. In *Tools for Teaching Logic*, volume 6680 of *Lecture Notes in Computer Science*, pages 62–69. Springer Berlin Heidelberg, 2011.

[8] João F. Ferreira. *Principles and Applications of Algorithmic Problem Solving*. PhD thesis, School of Computer Science, University of Nottingham, 2010.

[9] Daniel D. Garcia and David Ginat. Demystifying computing with magic. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 83–84, New York, NY, USA, 2012. ACM.

[10] Daniel D. Garcia and David Ginat. Demystifying computing with magic, continued. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 207–208, New York, NY, USA, 2013. ACM.

[11] Gerald Kruse. "Magic numbers" approach to introducing binary number representation in CSO. *SIGCSE Bull.*, 35(3):272–272, 2003.

[12] Anany Levitin and Mary-Angela Papalaskari. Using puzzles in teaching algorithms. *SIGCSE Bull.*, 34(1):292–296, 2002.

[13] P. W. McOwan and P. Curzon. *The Magic of Computer Science*. 2008. Available from: `http://www.cs4fn.org/magic/downloads/cs4fnmagicbook1.pdf`. Last accessed: 17 Mar 2014.

[14] P. W. McOwan, P. Curzon, and J. Black. *The Magic of Computer Science II: Now we have your attention*. 2009. Available from: `http://www.cs4fn.org/magic/downloads/cs4fnmagicbook2.pdf`. Last accessed: 17 Mar 2014.

[15] Z. Michalewicz and M. Michalewicz. *Puzzle-based Learning: Introduction to Critical Thinking, Mathematics, and Problem Solving*. Hybrid Publishers, 1st edition, 2008.

[16] Paul Ramsden. *Learning to Teach in Higher Education*. Routledge, 2nd edition, 2003.

[17] Alison Shreeve, Shân Wareing, and Linda Drew. *Teaching for Quality Learning at University: What the Student does (Society for Research Into Higher Education)*, chapter Key aspects of teaching and learning in the visual arts. In [3], 4th edition, 2011.